BAB 3 ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

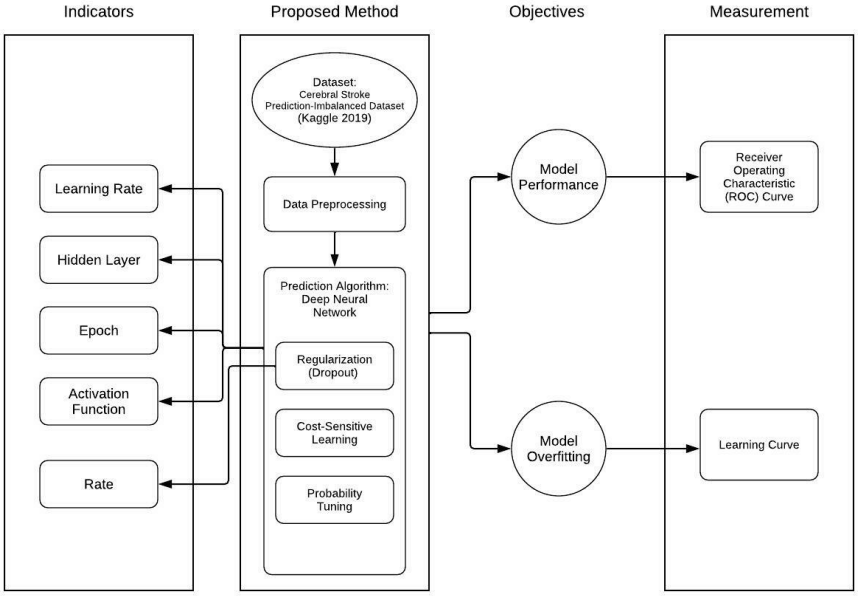
Seperti yang sudah dijelaskan pada Bab 1, banyak penelitian sebelumnya yang sudah menggunakan beberapa metode *machine learning* untuk prediksi stroke, namun metode tersebut tentu saja memiliki banyak kelemahan yang dapat diatasi oleh metode *deep learning*, khususnya *Deep Neural Network* (DNN). *Dataset* dalam penelitian ini bersifat *imbalanced*, sehingga akan digunakan metode tambahan untuk *handling imbalance class*, yaitu *cost-sensitive learning* [9] dan *probability tuning*. Metode DNN dapat mendeteksi hubungan yang kompleks dan mendapatkan akurasi yang lebih tinggi, dibandingkan metode *machine learning* tradisional, namun DNN juga memiliki kelemahan, yaitu mudah mengalami *overfitting* [7] dan waktu *training* yang lambat [4] [8]. Untuk mengatasi kelemahan DNN, maka penggunaan *regularization dropout* [10] [11] dan pencarian kombinasi *hyperparameter* terbaik dinilai dapat mengatasi permasalahan tersebut.

Penelitian ini akan membangun, menguji, dan membandingkan antara model DNN biasa dan model DNN dengan menggunakan *dropout*, serta melihat seberapa besar pengaruh *overfitting* terhadap *dataset tabular* dalam kasus prediksi orang terkena penyakit stroke dengan menggunakan ROC *curve*.

Input untuk sistem ini adalah data penunjang yang sudah ditentukan, seperti *gender*, *age*, *hypertension*, *heart_disease*, *avg_glucose_level*, *bmi*, dan *smoking_status*. *Output* sistem ini berupa hasil prediksi bahwa seseorang menderita stroke atau tidak.

3.2 Kerangka Pemikiran

Pada Gambar 3.1 diberikan gambar mengenai kerangka pemikiran dalam penelitian ini.



Gambar 3.1 Kerangka Pemikiran

Berikut akan dijelaskan setiap bagian yang ada pada gambar 3.1

1. *Indicators* adalah variabel-variabel yang digunakan dan akan memengaruhi hasil akhir. Indikator yang digunakan dalam penelitian ini adalah sebagai berikut.
   (a) *Learning rate*, berfungsi untuk mengatur seberapa besar model melakukan *update weight*. Semakin kecil model dapat konvergen, tetapi diperlukan *epoch* yang besar, otomatis waktunya akan semakin lama. Jika terlalu besar, model tidak dapat konvergen. *Learning rate* yang akan digunakan dalam penelitian ini adalah 0,1 dan 0,01, karena dalam Penelitian [7], *learning rate* yang digunakan sebesar 0,1 dan mendapatkan hasil terbaik, sedangkan pada Buku [13] dijelaskan bahwa learning rate yang *terlalu kecil* dapat konvergen namun membutuhkan waktu yang lama. Jika *learning rate* terlalu besar, model tidak dapat konvergen. Oleh sebab itu, digunakan *learning rate* yang lebih kecil, yaitu 0,01.
   (b) *Hidden layer*, berfungsi untuk pola-pola yang tidak terlihat di *neural network*. Semakin banyak, maka akan semakin lambat, tetapi bisa menemukan pola yang kompleks. *Hidden layer* yang akan digunakan dalam penelitian ini adalah 5, 10, dan 20, Dalam Penelitian [4] menggunakan *hidden layer* hanya sebesar 5 dan mendapatkan akurasi yang cukup memuaskan, yaitu 86,42%, sedangkan dalam Penelitian [7] menggunakan beragam *hidden layer*, mulai dari 10 sampai 50, namun penulis akan mencoba *hidden layer* 10 dan 20, dikarenakan hasilnya sudah sangat

memuaskan, yaitu pada *hiddden layer* 10 mendapatkan ROC tertinggi 0,97% dan akurasi 99,5% sendangkan pada *hidden layer* 20, model mendapatkan ROC tertinggi 99% dan akurasi 99,3%

(c) *Epoch*, merupakan iterasi 1 siklus program selesai dijalankan. Semakin besar semakin bisa meningkatkan akurasi, namun akan semakin lama, dan dapat menyebabkan *overfitting*. *Epoch* yang akan digunakan dalam penelitian ini adalah sebesar 10, 50, dan 100, karena pada Penelitian [8] memakai *epoch* sebesar 50 dan mendapatkan akurasi 84,03%, sedangkan pada Penelitian [7] menggunakan beragam *epoch*, yaitu dari 100 hingga 1000, tetapi hasil dari *epoch* 100 sendiri sudah baik, yaitu sudah mencapai 93,9%. Disisi lain, penulis ingin membuktikan apakah model tidak mendapat akurasi yang baik jika *epoch* diperkecil. Oleh sebab itu, digunakan *epoch* 10.

(d) *Activation function*, berfungsi untuk menentukan *output* dari *neural network*, dengan *range* 0 sampai 1, -1 sampai 1, 0 sampai x. Nilai *range* tergantung dari masing-masing *activation function*. *Activation function* yang akan digunakan dalam penelitian ini adalah ReLu dan Tanh pada *input* dan *hidden layer*, dan sigmoid pada *output layer*. Ketiga *activation function* ini digunakan karena dalam Buku [14], disebutkan bahwa ketiga *activation function* ini merupakan *activation function* yang populer dipakai dan dalam Penelitian [7], *activation function* yang digunakan adalah ReLu, Tanh, dan sigmoid. Semua percobaannya menghasilkan ROC curve paling kecil 90% dan *sensitivity* atau *recall* paling kecil adalah 83,8.%

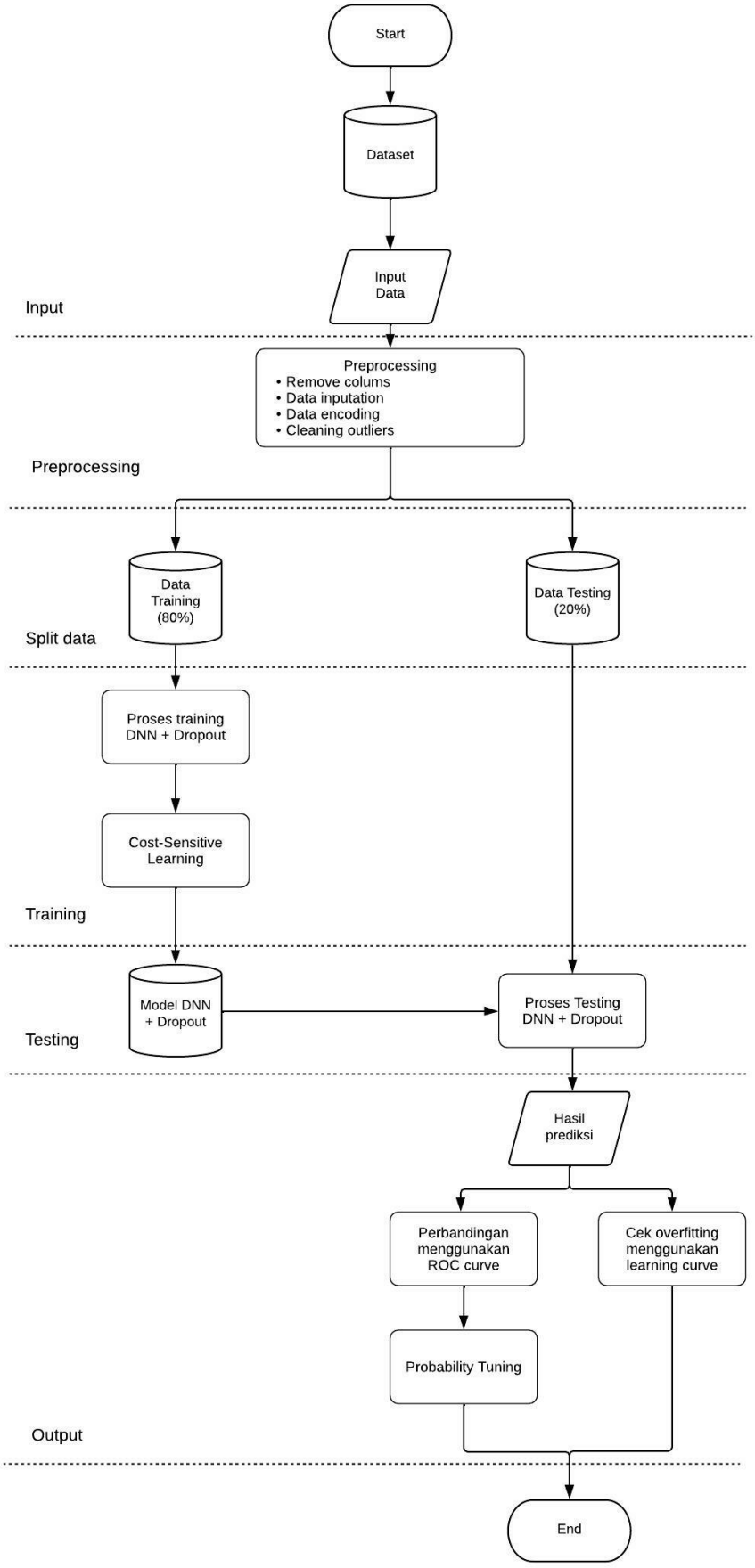
(e) *Rate*, merupakan probabilitas mempertahankan unit. Probabilitas = 1 artinya tidak ada *dropout*, dan semakin kecil nilai probabilitasnya, semakin banyak *neuron* yang di*dropout*. Semakin kecil nilai probabilitas, artinya semakin membutuhkan banyak *neuron* yang otomatis akan memperlambat *training* dan hasilnya akan cenderung *underfitting*. *Rate* yang akan digunakan dalam penelitian ini adalah 0,1 di antara *input layer* dan *hidden layer*, 0,5, dan 0,8 di antara *hidden layer* dan *output layer*.

2. *Proposed Method* adalah bagian yang menjelaskan proses penelitian dari awal hingga akhir. Proses pertama kali adalah melakukan *preprocessing*, yaitu dengan menghapus *outliers*, melakukan *data imputation* dikarenakan terdapat variabel yang memiliki *missing value*, dan *handling imbalance class*. Setelah itu akan dibuat model dengan arsitektur *Deep Neural Network* yang ditambah dengan metode *regularization dropout* untuk mencegah *overfitting* pada model.
3. *Objectives* adalah bagian yang menjelaskan acuan pengukuran. Penelitian ini menggunakann acuan performa dan model *overfitting*.

4. *Measurement* adalah bagian yang menjelaskan ukuran yang dipakai pada bagian *objectives*. Penelitian ini menggunakan *Receiver Operating Characteristic* (ROC) *curve*.

3.3 Urutan Proses Global

Pada Gambar 3.2 diberikan *flowchart* mengenai urutan proses dalam penelitian ini.

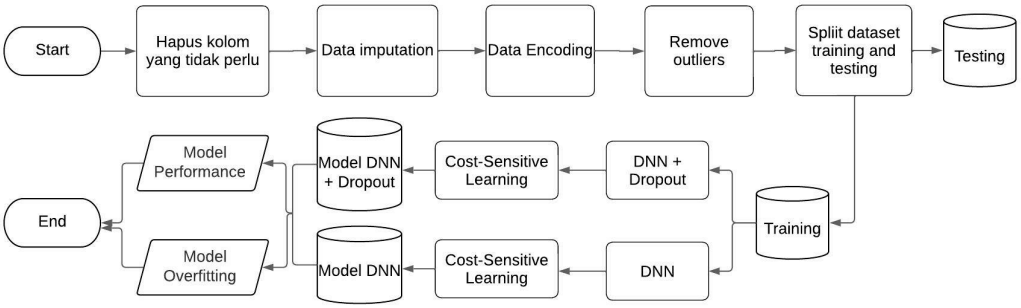

Start
Dataset
Input Data
Input
Preprocessing
• Remove colums
• Data inputation
• Data encoding
• Cleaning outliers
Preprocessing
Data Training (80%)
Data Testing (20%)
Split data
Proses training DNN + Dropout
Cost-Sensitive Learning
Training
Model DNN + Dropout
Proses Testing DNN + Dropout
Testing
Hasil prediksi
Perbandingan menggunakan ROC curve
Cek overfitting menggunakan learning curve
Probability Tuning
Output
End

Gambar 3.2 Urutan Proses Global

Model prediksi stroke ini dibangun menggunakan algoritma DNN dengan menggunakan *regularization dropout*. Setelah dilakukan *training*, model diharapkan dapat memprediksi kemungkinan orang yang terkena stroke dengan akurat dan cepat. Seperti pada Gambar 3.2, dimulai dari *preprocessing dataset*, lalu melakukan *training* dan *testing* untuk membuat model. Setelah model selesai dibuat, maka akan dilakukan proses *testing* dengan data yang bersumber dari data *testing*. Jika proses *testing* selesai, maka sistem akan menghasilkan prediksi, yang akan dicek performanya menggunakan ROC *curve*.

3.3.1 Proses *Training*

Pada penelitian ini, proses *training* model digambarkan pada gambar 3.3.



Gambar 3.3 *Flowchart* proses *training*

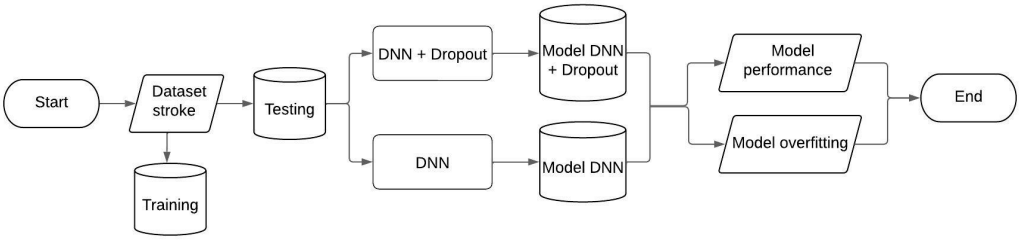
1. Dataset yang digunakan terdiri dari 43.400 data yang memiliki 12 variabel, yaitu *id*, *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, *smoking_status*, dan *stroke*.
2. Selanjutnya, akan dilakukan penghapusan terhadap variabel *id*.
3. *Data imputation* dilakukan menggunakan *mean imputation* dan mengganti nilai *nan* dengan kategori *unknown*.
4. Dilakukan data *encoding* yaitu mengganti isi dari variabel kategorikal menjadi angka, yang berguna agar model dapat fit pada saat membuat model Deep Neural Network (DNN).
5. Dilakukan penghapusan terhadap *outliers*.
6. Membagi *dataset* sebesar 80% untuk proses *training Deep Neural Network* (DNN). Pembagian dataset sebanyak 80% dipilih karena menurut Buku [13], pembagian data *training* yang paling umum adalah 80%.
7. Akan menghasilkan 2 model, yaitu model DNN biasa yang akan dibandingkan dengan model DNN yang ditambahkan *dropout*.
8. Dikarenakan kelas stroke dan tidak terkena stroke tidak seimbang, maka akan

dilakukan teknik *cost-sensitive learning* dan *probability tuning*.

9. Terakhir, akan diukur performa dari masing-masing model menggunakan ROC *curve* dan dibandingkan menggunakan AUC. Pengukuran mengenai *overfitting* akan dilakukan menggunakan *learning curves*.

3.3.2 Proses *Testing*

Pada penelitian ini, proses *testing* model digambarkan pada gambar 3.4.



Gambar 3.4 *Flowchart* proses *testing*

1. *Input* yang digunakan terdiri dari 43.400 data yang memiliki 10 fitur, yaitu *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, dan *smoking_status*.
2. Membagi *dataset* sebesar 20% untuk proses *training Deep Neural Network* (DNN). Pembagian dataset *testing* sebanyak 20% dipilih karena menurut Buku [13], pembagian data *training* yang paling umum adalah 20%, namun jika memiliki sampai 10 juta, maka cukup menggunakan 1% untuk data *testing*.
3. Akan menghasilkan 2 model, yaitu model DNN biasa yang akan dibandingkan dengan model DNN yang ditambahkan *dropout*.
4. Terakhir, akan diukur performa dari masing-masing model menggunakan ROC *curve* dan dibandingkan menggunakan AUC dan model *overfitting* diukur menggunakan *learning curves*.

3.4 Analisis Manual

Pada bagian ini akan dijelaskan analisis tahapaan proses yang dilakukan dalam sistem.

3.4.1 Dataset

Dataset berjumlah 43.400 data dan memiliki 12 variabel, yaitu sebagai berikut.

1. *id*: id sebagai angka unik.
2. *gender*: jenis kelamin (*male*, *female*, *other*).
3. *age*: umur pasien.

4. *hypertension*: hipertensi seseorang (0 jika tidak memiliki hipertensi, 1 jika memiliki hipertensi)
5. *heart_disease*: penyakit jantung (0 jika tidak memiliki penyakit jantung, 1 jika memiliki penyakit jantung)
6. *ever_married*: status pernikahan (*no* dan *yes*)
7. *work_type*: status pekerjaan (*children*, *govt_jov*, *never_worked*, *private* or *self-employed*)
8. *residence_type*: tempat tinggal seseorang (*rural* atau *urban*)
9. *avg_glucose_level*: tingkat glukosa rata-rata dalam darah
10. *bmi*: *body mass index*
11. *smoking_status*: status merokok pada seseorang (*formerly smoked*, *never smoked*, *smokes*)
12. *stroke*: status menderita stroke pada seseorang 0 jika tidak menderita stroke, 1 jika menderita stroke)

Dataset yang digunakan dalam penelitian ini bersifat imbalanced class, yang dibuktikan dengan perhitungan rasio yang dipakai dalam perbandingan dan persentase sesuai penjelasan pada Bab 2 sebagai berikut:

1. 783 : 42.617 = 1 : 54,42.
2. (783/42.617) * 100% = 1,84%.

Hasil rasio diatas menunjukkan bahwa dataset yang digunakan termaksud *imbalanced class moderate*.

Terdapat berbagai macam teknik untuk mengatasi *imbalanced class*, seperti yang sudah dijelaskan pada bagian 2.1.3. Dalam kasus prediksi penyakit stroke, metode *data sampling*, baik *oversampling* maupun *undersampling* tidak cocok digunakan. Hal ini dikarenakan jika kasus penyakit dilakukan *oversampling*, maka artinya sudah dilakukan penambahan data selain data aslinya, yang menyebabkan datanya sudah tidak akurat lagi, sedangkan jika dilakukan *undersampling*, artinya akan dilakukan penghapusan data yang menyebabkan datanya menjadi sangat sedikit mengikuti kelas minoritas, sehingga akan digunakan teknik cost-sensitive learning dan probability tuning untuk mengatasi *imbalanced class*.

3.4.2 *Preprocessing*

Pada tahap ini dilakukan *preprocessing* terhadap *dataset* sebelum dilakukan *training* dan *testing* pada metode Deep Neural Network (DNN) dan *dropout*.

3.4.2.1 Menghapus Kolom *id*

Pada tahap ini dilakukan penghapusan kolom *id* karena *id* hanya identitas angka pasien, yang tidak ada hubungannya dengan prediksi penyakit stroke. Gambar 3.5 merupakan kode penghapusan kolom pada Python dan Gambar 3.6 merupakan contoh *dataset* setelah dilakukan penghapusan beberapa kolom.

```
1 df.drop(['id'], axis=1, inplace=True)
2 df
```

Gambar 3.5 Penghapusan kolom menggunakan kode Python

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
1	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	0
3	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	No	children	Urban	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	Yes	Govt_job	Urban	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	Yes	Private	Urban	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	Yes	Private	Urban	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	Yes	Private	Urban	79.48	20.6	never smoked	0

43400 rows × 11 columns

Gambar 3.6 Penghapusan kolom *id*

3.4.2.2 Data *Imputation*

Pada fitur *bmi* terdapat *missing values* sedangkan pada fitur *smoking_status* terdapat *NaN*. Oleh karena itu, untuk fitur *bmi* dilakukan data *imputation* dengan cara *mean imputation* dan untuk fitur *smoking_status* dilakukan dengan cara mengganti *NaN* dengan *unknown*. Gambar 3.7 merupakan kode data *imputation* pada Python dan Gambar 3.8 merupakan contoh *dataset* setelah dilakukan data *imputation*.

```
df.isnull().sum()
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                 1462
smoking_status     13292
stroke                 0
dtype: int64
```

```
df['bmi'].fillna(df['bmi'].mean(), inplace=True)
df['smoking_status'] = df['smoking_status'].replace(np.nan, 'unknown')
```

```
df.isnull().sum()
```

```
gender             0
age                0
hypertension       0
heart_disease      0
ever_married       0
work_type          0
Residence_type     0
avg_glucose_level  0
bmi                0
smoking_status     0
stroke             0
dtype: int64
```

Gambar 3.7 Data *imputation* menggunakan kode Python

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	No	children	Rural	95.12	18.0	unknown	0
1	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	Female	8.0	0	0	No	Private	Urban	110.89	17.6	unknown	0
3	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	unknown	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	No	children	Urban	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	Yes	Govt_job	Urban	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	Yes	Private	Urban	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	Yes	Private	Urban	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	Yes	Private	Urban	79.48	20.6	never smoked	0

43400 rows × 11 columns

Gambar 3.8 Data *imputation*

3.4.2.3 Data *Encoding*

Pada tahap ini, fitur kategorikal, seperti *gender*, *ever_married*, *work_type*, *residence_type*, dan *smoking_status* akan dilakukan *encoding*, yaitu dengan mengganti isinya menjadi angka, dengan rincian sebagai berikut:

1. Fitur *gender* yang berisi *male*, *female*, dan *other* diganti menjadi 0, 1, dan 2.
2. Fitur *ever_married* yang berisi *no* dan *yes* diganti menjadi 0 dan 1.
3. Fitur *work_type* yang berisi *never_worked*, *children*, *govt_job*, *private*, dan *self-employed* diganti menjadi 0, 1, 2, 3, dan 4.
4. Fitur *residence_type* yang berisi *rural* dan *urban* diganti menjadi 0 dan 1.
5. Fitur *smoking_status* yang berisi *unknown*, *never smoked*, *formerly smoked*, *smokes*

diganti menjadi 0, 1, 2, dan 3.

Hal ini dilakukan agar model dapat *fit* pada saat membuat model Deep Neural Network (DNN). Gambar 3.9 merupakan kode data *encoding* pada Python dan Gambar 3.10 merupakan contoh dataset setelah dilakukan data *encoding*.

```
df['gender'].replace(['Male', 'Female', 'Other'],[0, 1, 2], inplace=True)
df['ever_married'].replace(['No', 'Yes'],[0, 1], inplace=True)
df['work_type'].replace(['Never_worked', 'children', 'Govt_job', 'Private', 'self-employed'],[0, 1, 2, 3, 4], inplace=True)
df['Residence_type'].replace(['Rural', 'Urban'],[0, 1], inplace=True)
df['smoking_status'].replace(['unknown','never smoked', 'formerly smoked', 'smokes'],[0, 1, 2,3], inplace=True)
```

Gambar 3.9 *Encoding* menggunakan kode Python

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	3.0	0	0	0	1	0	95.12	18.0	0	0
1	0	58.0	1	0	1	3	1	87.96	39.2	1	0
2	1	8.0	0	0	0	3	1	110.89	17.6	0	0
3	1	70.0	0	0	1	3	0	69.04	35.9	2	0
4	0	14.0	0	0	0	0	0	161.28	19.1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	1	10.0	0	0	0	1	1	58.64	20.4	1	0
43396	1	56.0	0	0	1	2	1	213.61	55.4	2	0
43397	1	82.0	1	0	1	3	1	91.94	28.9	2	0
43398	0	40.0	0	0	1	3	1	99.16	33.2	1	0
43399	1	82.0	0	0	1	3	1	79.48	20.6	1	0

43400 rows × 11 columns

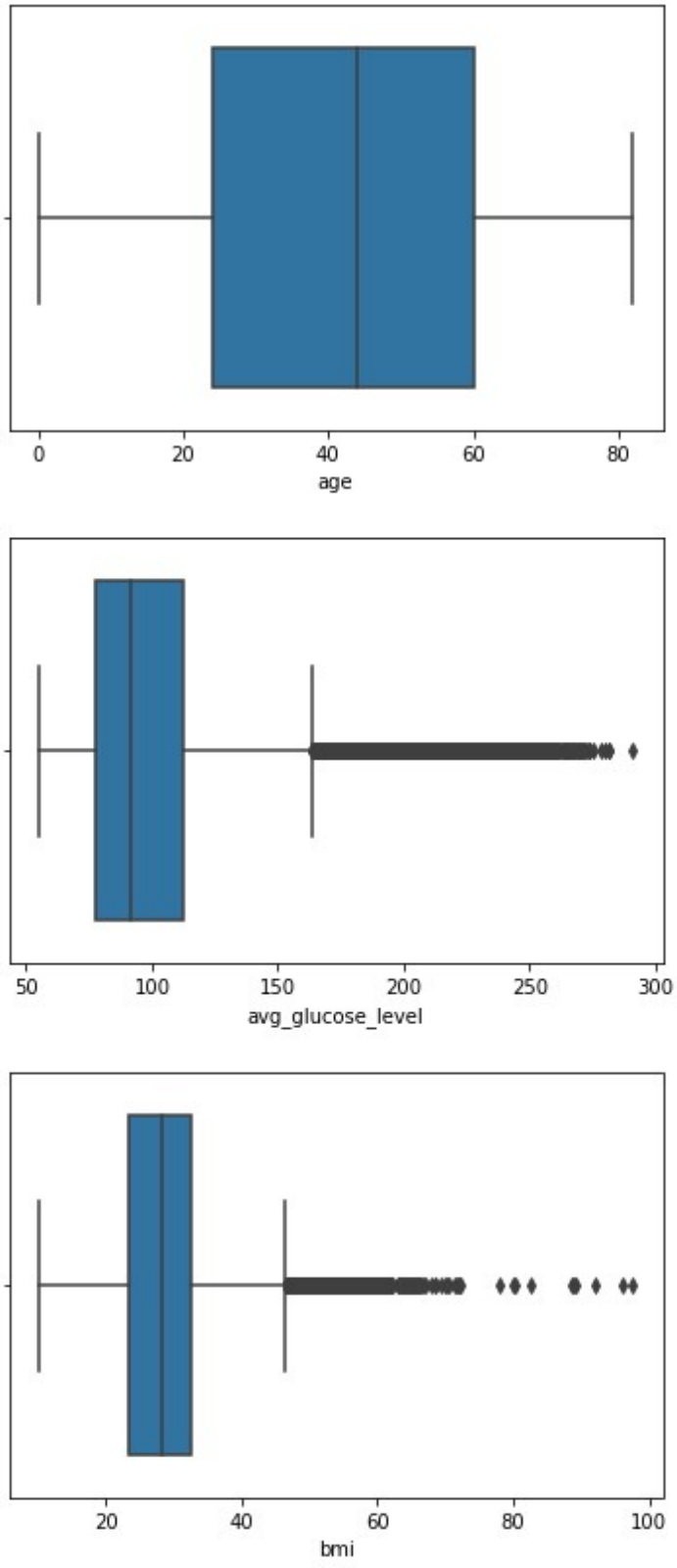
Gambar 3.10 *Encoding* pada fitur *gender*, *ever_married*, *work_type*, *residence_type*, dan *smoking_status*

3.4.2.4 *Cleaning Outliers*

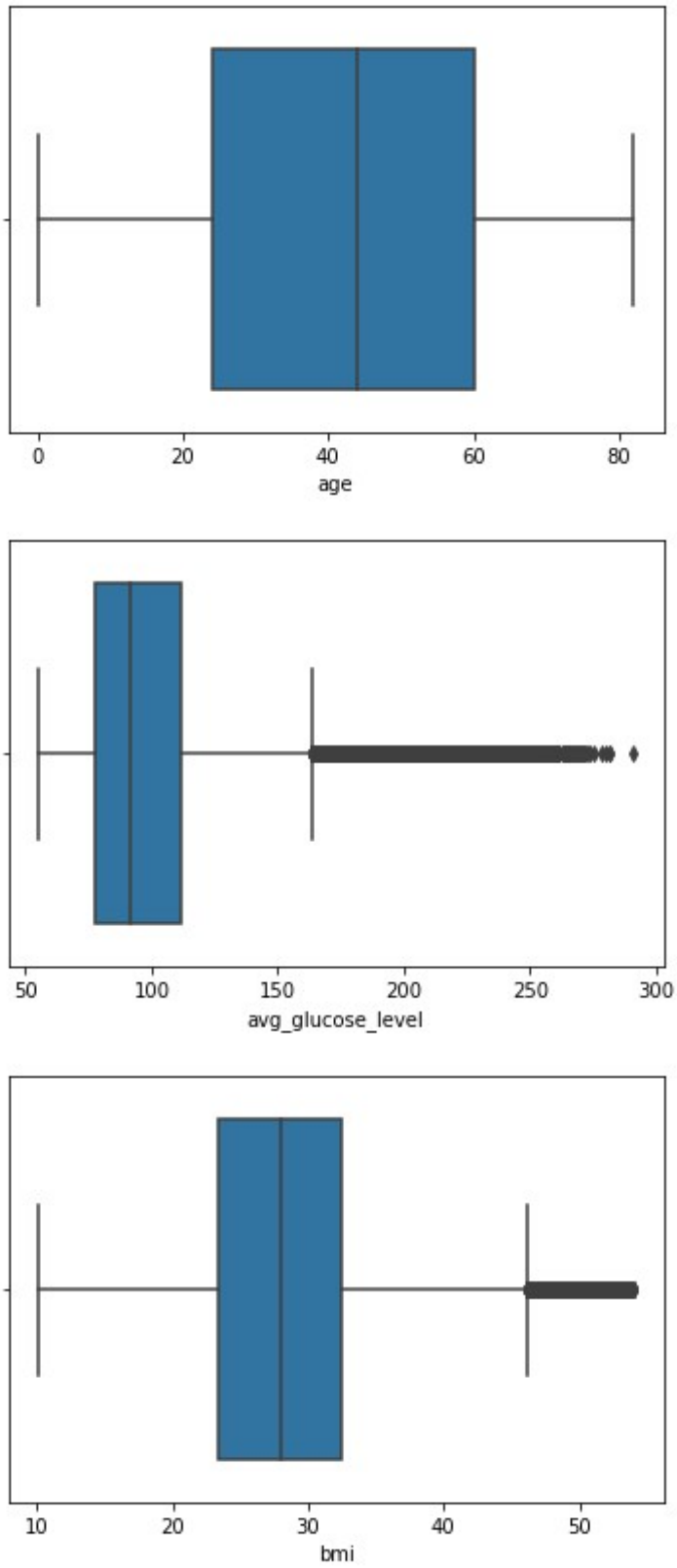
Dataset akan dilakukan pengecekan terhadap *outliers* menggunakan *boxplot*. Fitur *age* tidak memiliki *outliers*. *Outliers* pada fitur*avg glucose level* tidak akan dihapus, karena memungkinkan memiliki kadar gula darah lebih dari 300 mg/dl [29]. *Outliers* pada fitur bmi akan dihapus dan data yang diambil hanya dari data awal sampai 54, dikarenakan sesuai penjelasan pada Bagian 2.3.1, seseorang yang memiliki bmi 54 sudah termaksud *extreme obesity* [28] sehingga bmi lebih dari 54 sudah tidak bisa dipertanggungjawabkan kebenarannya. Gambar 3.11 merupakan kode *cleaning* outliers pada Python, sedangkan Gambar 3.12 merupakan *boxplot* sebelum dilakukan data *cleaning* dan Gambar 3.13 merupakan *boxplot* sesudah dilakukan data *cleaning*.

```
df = df[(df['bmi'] <= 54 )]
df.describe()
```

Gambar 3.11 *Cleaning outliers* menggunakan kode Python



Gambar 3.12 *Boxplot* sebelum dilakukan data *cleaning*



Gambar 3.13 *Boxplot* sesudah dilakukan data *cleaning*

3.4.3 *Split Dataset* untuk *Training* dan *Testing*

Pada tahap ini dilakukan pembagian *dataset* untuk proses *training* dan *testing* yang akan digunakan untuk membuat model *Deep Neural Network*. Pembagian *dataset* adalah 80% untuk data *training* dan 20% untuk data *testing*.

3.5 Perhitungan *Deep Neural Network*

Arsitektur yang akan digunakan dalam *neural network* dibagi menjadi 3 *layer*, yaitu:

1. *S*atu buah input layer yang terdiri dari 10 *neuron* yang merepresentasikan 10 fitur dan menggunakan *activation function* tanh.
2. *S*atu buah hidden layer yang terdiri dari 10 *neuron* yang menggunakan *activation function* tanh.
3. Satu buah *output layer* yang terdiri dari 1 *neuron* dan menggunakan *activation function sigmoid*.

Gambar arsitektur *Deep Neural Network* dapat dilihat pada Gambar 2.1.

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 120 data secara *random* dengan rentang tertentu.

Inisialisasi Data:
Jumlah *input layer* = 1
Jumlah *neuron* pada *input layer* = 10
Activation function pada *input layer* = tanh

Jumlah *hidden layer* = 1
Jumlah *neuron* pada *hidden layer* = 10
Activation function pada *hidden layer* = tanh

Jumlah *output layer* = 1
Jumlah *neuron* pada *output layer* = 1
Activation function pada *output layer* = *sigmoid*

Rentang *weight* = -0,31622776601683794 sampai dengan 0,31622776601683794

Bias = 0

Input (weight setiap fitur yang ada):

$x1$ = -0,02835948 $x2$ = -0,08527658 $x3$ = 0,26398902 $x4$ = -0,02807259 $x5$ = 0,04415979 $x6$ = -0,15166705 $x7$ = -0,18228751 $x8$ = 0,3134241 $x9$ = -0,00786456 $x10$ = 0,17467744

Hidden layer:

$w_{k1}j_1$ = -0,08154228 $w_{k2}j_1$ = 0,30040546 $w_{k3}j_1$ = 0,24854657 $w_{k4}j_1$ = 0,01691013 $w_{k5}j_1$ = -0,18283772 $w_{k6}j_1$ = 0,15720913 $w_{k7}j_1$ = 0,08201578 $w_{k8}j_1$ = -0,19598779 $w_{k9}j_1$ = 0,26268099 $w_{k10}j_1$ = 0,13676267

$w_{k1}j_2$ = 0,21293817 $w_{k2}j_2$ = -0,25220019 $w_{k3}j_2$ = 0,29390853 $w_{k4}j_2$ = -0,18997839 $w_{k5}j_2$ = -0,14565171 $w_{k6}j_2$ = 0,19445793 $w_{k7}j_2$ = -0,17048571 $w_{k8}j_2$ = -0,13914967 $w_{k9}j_2$ = -0,11090133 $w_{k10}j_2$ = 0,09844929

$w_{k1}j_3$ = -0,22548086 $w_{k2}j_3$ = 0,14510369 $w_{k3}j_3$ = 0,22841273 $w_{k4}j_3$ = -0,22168199 $w_{k5}j_3$ = 0,05429651 $w_{k6}j_3$ = -0,22866022 $w_{k7}j_3$ = -0,24450572 $w_{k8}j_3$ = -0,24502611 $w_{k9}j_3$ = -0,30133026 $w_{k10}j_3$ = -0,306759

$w_{k1}j_4$ = 0,03463625 $w_{k2}j_4$ = -0,00885326 $w_{k3}j_4$ = -0,22394442 $w_{k4}j_4$ = -0,06561425 $w_{k5}j_4$ = 0,07484731 $w_{k6}j_4$ = 0,01486269 $w_{k7}j_4$ = -0,10394584 $w_{k8}j_4$ = 0,17709923 $w_{k9}j_4$ = 0,07621706 $w_{k10}j_4$ = -0,09844928

$w_{k1}j_5$ = -0,12021486 $w_{k2}j_5$ = -0,07829361 $w_{k3}j_5$ = -0,12997544 $w_{k4}j_5$ = 0,21279237 $w_{k5}j_5$ = 0,20977342 $w_{k6}j_5$ = -0,22495591 $w_{k7}j_5$ = -0,00190682 $w_{k8}j_5$ = -0,22943178 $w_{k9}j_5$ = 0,27366045 $w_{k10}j_5$ = -0,20595517

$w_{k1}j_6$ = -0,00716274 $w_{k2}j_6$ = -0,05448304 $w_{k3}j_6$ = 0,06476339 $w_{k4}j_6$ = 0,03532464 $w_{k5}j_6$ = 0,09245855 $w_{k6}j_6$ = 0,1695172 $w_{k7}j_6$ = 0,23680192 $w_{k8}j_6$ = 0,04477918 $w_{k9}j_6$ = 0,24950235 $w_{k10}j_6$ = -0,30254995

$w_{k1}j_7$ = -0,14745197 $w_{k2}j_7$ = -0,08891625 $w_{k3}j_7$ = 0,22152022 $w_{k4}j_7$ = -0,0214877 $w_{k5}j_7$ = -0,12194666 $w_{k6}j_7$ = -0,05346982 $w_{k7}j_7$ = 0,26132289 $w_{k8}j_7$ = 0,26464311 $w_{k9}j_7$ = 0,29086091 $w_{k10}j_7$ = 0,07891517

$w_{k1}j_8$ = 0,30138627 $w_{k2}j_8$ = -0,17017802 $w_{k3}j_8$ = 0,0968307 $w_{k4}j_8$ = 0,26289768 $w_{k5}j_8$ = 0,01635267 $w_{k6}j_8$ = -0,06321122 $w_{k7}j_8$ = 0,1968099 $w_{k8}j_8$ = -0,14343715 $w_{k9}j_8$ = -0,01907697 $w_{k10}j_8$ = -0,24814027

$w_{k1}j_9$ = 0,30255215 $w_{k2}j_9$ = -0,08199351 $w_{k3}j_9$ = 0,17765137 $w_{k4}j_9$ = 0,07950517 $w_{k5}j_9$ = 0,15693231 $w_{k6}j_9$ = -0,06841941 $w_{k7}j_9$ = -0,01554272 $w_{k8}j_9$ = 0,21543019 $w_{k9}j_9$ = -0,19608282 $w_{k10}j_9$ = 0,27772592

$w_{k1}j_{10}$ = -0,27114336 $w_{k2}j_{10}$ = -0,12175201 $w_{k3}j_{10}$ = 0,03013474 $w_{k4}j_{10}$ = -0,10994942 $w_{k5}j_{10}$ = 0,10416485 $w_{k6}j_{10}$ = 0,21983722 $w_{k7}j_{10}$ = 0,296758 $w_{k8}j_{10}$ = 0,10953304 $w_{k9}j_{10}$ = -0,06145034 $w_{k10}j_{10}$ = 0,27154544

Output layer:

$w_{k1}j_{11}$ = -0,14147132 $w_{k2}j_{11}$ = 0,05554158 $w_{k3}j_{11}$ = -0,27054289 $w_{k4}j_{11}$ = -0,03277796 $w_{k5}j_{11}$ = -0,10409342 $w_{k6}j_{11}$ = -0,24334135 $w_{k7}j_{11}$ = -0,06665223 $w_{k8}j_{11}$ = 0,01262219 $w_{k9}j_{11}$ = 0,18286435 $w_{k10}j_{11}$ = 0,16453867

Berikut ini merupakan perhitungan terhadap *hidden layer* dengan menggunakan Persamaan 2.1 dan 2.3:

$h1 = ((w_{k1}j_1 * x1) + (w_{k2}j_1 * x2) + (w_{k3}j_1 * x3) + (w_{k4}j_1 * x4) + (w_{k5}j_1 * x5) + (w_{k6}j_1 * x6) + (w_{k7}j_1 * x7) + (w_{k8}j_1 * x8) + (w_{k9}j_1 * x9) + (w_{k10}j_1 * x10)) + b$

$h1$ = ((-0,08154228 * -0,02835948) + (-0,02835948 * -0,08527658) + (0,24854657 * 0,26398902) + (0,01691013 * -0,02807259) + (-0,18283772 * 0,04415979) + (0,15720913 * -0,15166705) + (0,08201578 * -0,18228751) + (-0,19598779 * 0,3134241) + (0,26268099 * -0,00786456) + (0,13676267 * 0,17467744)) + 0

$h1$ = -0,016602036215524193

$h2 = ((w_{k1}j_2 * x1) + (w_{k2}j_2 * x2) + (w_{k3}j_2 * x3) + (w_{k4}j_2 * x4) + (w_{k5}j_2 * x5) + (w_{k6}j_2 * x6) + (w_{k7}j_2 * x7) + (w_{k8}j_2 * x8) + (w_{k9}j_2 * x9) + (w_{k10}j_2 * x10)) + b$

$h2$ = ((0,21293817 * -0,02835948) + (-0,25220019 * -0,08527658) + (0,29390853 * 0,26398902) + (-0,18997839 * -0,02807259) + (-0,14565171 * 0,04415979) + (0,19445793 * -0,15166705) + (-0,17048571 * -0,18228751) + (-0,13914967 * 0,3134241) + (-0,11090133 * -0,00786456) + (0,09844929 * 0,17467744)) + 0

$h2$ = 0,06799857023423941

$h3 = ((w_{k1}j_3 * x1) + (w_{k2}j_3 * x2) + (w_{k3}j_3 * x3) + (w_{k4}j_3 * x4) + (w_{k5}j_3 * x5) + (w_{k6}j_3 * x6) + (w_{k7}j_3 * x7) + (w_{k8}j_3 * x8) + (w_{k9}j_3 * x9) + (w_{k10}j_3 * x10)) + b$

$h3$ = ((-0,22548086 * -0,02835948) + (0,14510369 * -0,08527658) + (0,22841273 * 0,26398902) + (-0,22168199 * -0,02807259) + (0,05429651 * 0,04415979) + (-0,22866022 * -0,15166705) + (-0,24450572 * -0,18228751) + (0,24502611 * 0,3134241) + (-0,30133026 * -0,00786456) + (-0,306759 * 0,17467744)) + 0

$h3$ = 0,16777353734936898

$h4 = ((w_{k1}j_4 * x1) + (w_{k2}j_4 * x2) + (w_{k3}j_4 * x3) + (w_{k4}j_4 * x4) + (w_{k5}j_4 * x5) + (w_{k6}j_4 * x6) + (w_{k7}j_4 * x7) + (w_{k8}j_4 * x8) + (w_{k9}j_4 * x9) + (w_{k10}j_4 * x10)) + b$

$h4$ = ((0,03463625 * -0,02835948) + (-0,00885326 * -0,08527658) + (-0,22394442 * 0,26398902) + (-0,06561425 * -0,02807259) + (0,07484731 * 0,04415979) + (0,01486269 * -0,15166705) + (-0,10394584 * -0,18228751) + (0,17709923 * 0,3134241) + (0,07621706 * -0,00786456) + (0,09844928 * 0,17467744)) + 0

$h4$ = 0,03459951448869128

$h5 = ((w_{k1}j_5 * x1) + (w_{k2}j_5 * x2) + (w_{k3}j_5 * x3) + (w_{k4}j_5 * x4) + (w_{k5}j_5 * x5) + (w_{k6}j_5 * x6) + (w_{k7}j_5 * x7) + (w_{k8}j_5 * x8) + (w_{k9}j_5 * x9) + (w_{k10}j_5 * x10)) + b$

$h5$ = ((-0,12021486 * -0,02835948) + (-0,07829361 * -0,08527658) + (-0,12997544 * 0,26398902) + (0,21279237 * -0,02807259) + (0,20977342 * 0,04415979) + (-0,22495591 * -0,15166705) + (-0,00190682 * -0,18228751) + (-0,22943178 * 0,3134241) + (0,27366045 * -0,00786456) + (-0,20595517 * 0,17467744)) + 0

$h5$ = -0,09650773091582979

$h6 = ((w_{k1}j_6 * x1) + (w_{k2}j_6 * x2) + (w_{k3}j_6 * x3) + (w_{k4}j_6 * x4) + (w_{k5}j_6 * x5) + (w_{k6}j_6 * x6) + (w_{k7}j_6 * x7) + (w_{k8}j_6 * x8) + (w_{k9}j_6 * x9) + (w_{k10}j_6 * x10)) + b$

$h6 = ((-0,00716274 * -0,02835948) + (-0,05448304 * -0,08527658) + (0,06476339 * 0,26398902) + (0,03532464 * -0,02807259) + (0,09245855 * 0,04415979) + (0,1695172 * -0,15166705) + (0,23680192 * -0,18228751) + (0,04477918 * 0,3134241) + (0,24950235 * -0,00786456) + (-0,30254995 * 0,17467744)) + 0$

$h6 = -0,08461482998284209$

$h7 = ((w_{k1}j_7 * x1) + (w_{k2}j_7 * x2) + (w_{k3}j_7 * x3) + (w_{k4}j_7 * x4) + (w_{k5}j_7 * x5) + (w_{k6}j_7 * x6) + (w_{k7}j_7 * x7) + (w_{k8}j_7 * x8) + (w_{k9}j_7 * x9) + (w_{k10}j_7 * x10)) + b$

$h7 = ((-0,14745197 * -0,02835948) + (-0,08891625 * -0,08527658) + (0,22152022 * 0,26398902) + (0,0214877 * -0,02807259) + (-0,12194666 * 0,04415979) + (-0,05346982 * -0,15166705) + (0,26132289 * -0,18228751) + (0,26464311 * 0,3134241) + (0,29086091 * -0,00786456) + (0,07891517 * 0,17467744)) + 0$

$h7 = 0,1191711327063339$

$h8 = ((w_{k1}j_8 * x1) + (w_{k2}j_8 * x2) + (w_{k3}j_8 * x3) + (w_{k4}j_8 * x4) + (w_{k5}j_8 * x5) + (w_{k6}j_8 * x6) + (w_{k7}j_8 * x7) + (w_{k8}j_8 * x8) + (w_{k9}j_8 * x9) + (w_{k10}j_8 * x10)) + b$

$h8 = ((0,30138627 * -0,02835948) + (-0,17017802 * -0,08527658) + (0,0968307 * 0,26398902) + (0,26289768 * -0,02807259) + (0,01635267 * 0,04415979) + (-0,06321122 * -0,15166705) + (0,1968099 * -0,18228751) + (-0,14343715 * 0,3134241) + (-0,01907697 * -0,00786456) + (-0,24814027 * 0,17467744)) + 0$

$h8 = -0,0895708672147945$

$h9 = ((w_{k1}j_9 * x1) + (w_{k2}j_9 * x2) + (w_{k3}j_9 * x3) + (w_{k4}j_9 * x4) + (w_{k5}j_9 * x5) + (w_{k6}j_9 * x6) + (w_{k7}j_9 * x7) + (w_{k8}j_9 * x8) + (w_{k9}j_9 * x9) + (w_{k10}j_9 * x10)) + b$

$h9 = ((0,30255215 * -0,02835948) + (-0,08199351 * -0,08527658) + (0,17765137 * 0,26398902) + (0,07950517 * -0,02807259) + (0,15693231 * 0,04415979) + (-0,06841941 * -0,15166705) + (-0,01554272 * -0,18228751) + (-0,21543019 * 0,3134241) + (-0,19608282 * -0,00786456) + (0,27772592 * 0,17467744)) + 0$

$h9 = 0,0457518555709885$

$h10 = ((w_{k1}j_{10} * x1) + (w_{k2}j_{10} * x2) + (w_{k3}j_{10} * x3) + (w_{k4}j_{10} * x4) + (w_{k5}j_{10} * x5) + (w_{k6}j_{10} * x6) + (w_{k7}j_{10} * x7) + (w_{k8}j_{10} * x8) + (w_{k9}j_{10} * x9) + (w_{k10}j_{10} * x10)) + b$

$h10 = ((-0,27114336 * -0,02835948) + (-0,12175201 * -0,08527658) + (0,03013474 * 0,26398902) + (-0,10994942 * -0,02807259) + (0,10416485 * 0,04415979) + (0,21983722 * -0,15166705) + (0,296758 * -0,18228751) + (0,10953304 * 0,3134241) + (-0,06145034 * -0,00786456) + (0,27154544 * 0,17467744)) + 0$

$h10 = 0,028522880227219695$

Setelah mendapatkan hasil perhitungan terhadap *hidden layer*, selanjutnya diterapkan *activation function tanh* sesuai Persamaan 2.5.

$tanh\ (h1) = \frac{e^{h1} - e^{-h1}}{e^{h1} + e^{-h1}}$

$tanh\ (h1) = \frac{e^{-0,016602036215524193} - e^{--0,016602036215524193}}{e^{-0,016602036215524193} + e^{--0,016602036215524193}}$

$tanh\ (h1) = -0,0166005$

$tanh\ (h2) = \frac{e^{h2} - e^{-h2}}{e^{h2} + e^{-h2}}$

$tanh\ (h2) = \frac{e^{0,06799857023423941} - e^{-0,06799857023423941}}{e^{0,06799857023423941} + e^{-0,06799857023423941}}$

$tanh\ (h2) = 0,067894$

$tanh\ (h3) = \frac{e^{h3} - e^{-h3}}{e^{h3} + e^{-h3}}$

$tanh\ (h3) = \frac{e^{0,16777353734936898} - e^{-0,16777353734936898}}{e^{0,16777353734936898} + e^{-0,16777353734936898}}$

$tanh\ (h3) = 0,166217$

$tanh\ (h4) = \frac{e^{h4} - e^{-h4}}{e^{h4} + e^{-h4}}$

$tanh\ (h4) = \frac{e^{0,03459951448869128} - e^{-0,03459951448869128}}{e^{0,03459951448869128} + e^{-0,03459951448869128}}$

$tanh\ (h4) = 0,0345857$

$tanh\ (h5) = \frac{e^{h5} - e^{-h5}}{e^{h5} + e^{-h5}}$

$tanh\ (h5) = \frac{e^{-0,09650773091582979} - e^{--0,09650773091582979}}{e^{-0,09650773091582979} + e^{--0,09650773091582979}}$

$tanh\ (h5) = -0,0962092$

$tanh\ (h6) = \frac{e^{h6} - e^{-h6}}{e^{h6} + e^{-h6}}$

$tanh\ (h6) = \frac{e^{-0,08461482998284209} - e^{--0,08461482998284209}}{e^{-0,08461482998284209} + e^{--0,08461482998284209}}$

$tanh\ (h6) = -0,0844135$

$tanh\ (h7) = \frac{e^{h7} - e^{-h7}}{e^{h7} + e^{-h7}}$

$tanh\ (h7) = \frac{e^{0,1191711327063339} - e^{-0,1191711327063339}}{e^{0,1191711327063339} + e^{-0,1191711327063339}}$

$tanh\ (h7) = 0,11861$

$tanh\ (h8) = \frac{e^{h8} - e^{-h8}}{e^{h8} + e^{-h8}}$

$tanh\ (h8) = \frac{e^{-0,0895708672147945} - e^{--0,0895708672147945}}{e^{-0,0895708672147945} + e^{--0,0895708672147945}}$

$tanh\ (h8) = -0,0893321$

$tanh\ (h9) = \frac{e^{h9} - e^{-h9}}{e^{h9} + e^{-h9}}$

$tanh\ (h9) = \frac{e^{0,0457518555709885} - e^{-0,0457518555709885}}{e^{0,0457518555709885} + e^{-0,0457518555709885}}$

$tanh\ (h9) = 0,04572$

$tanh\ (h10) = \frac{e^{h10} - e^{-h10}}{e^{h10} + e^{-h10}}$

$tanh\ (h10) = \frac{e^{0,028522880227219695} - e^{-0,028522880227219695}}{e^{0,028522880227219695} + e^{-0,028522880227219695}}$

$tanh\ (h10) = 0,0285151$

Selanjutnya, dilakukan perhitungan terhadap *output layer* dengan menggunakan persamaan 2.1 dan 2.3:

$output = ((w_{k1} j_7 * \tanh(h1)) + (w_{k2} j_7 * \tanh(h2)) + (w_{k3} j_7 * \tanh(h3)) + (w_{k4} j_7 * \tanh(h4)) + (w_{k5} j_7 * \tanh(h5)) + (w_{k6} j_7 * \tanh(h6)) + (w_{k7} j_7 * \tanh(h7)) + (w_{k8} j_7 * \tanh(h8)) + (w_{k9} j_7 * \tanh(h9)) + (w_{k10} j_7 * \tanh(h10))) + b$

$output$ = ((-0,14147132 * -0,0166005) + (0,05554158 * 0,067894) + (-0,27054289 * 0,166217) + (-0,03277796 * 0,0345857) + (-0,10409342 * -0,0962092) + (-0,24334135 * -0,0844135) + (-0,06665223 * 0,11861) + (0,01262219 * -0,0893321) + (0,18286435 * 0,04572) + (0,16453867 * 0,0285151) + 0)

$output$ = -0,005407794875114999

Langkah terakhir, adalah melakukan perhitungan dengan *activation function sigmoid* pada *output layer* dengan menggunakakan persamaan 2.4 untuk mendapatkan hasil prediksi antara 0 (tidak stroke) atau 1 (stroke):

$\sigma(z) = = 1/(1 + exp(-z))$

$\sigma(-0,005407794875114999)$ = 1 / (1 + exp(- -0,005407794875114999))

$\sigma(-0,005407794875114999)$ = 1 / (1 + 1,005422443391307)

$\sigma(-0,005407794875114999)$ = 1 / 2,005422443391307

$\sigma(-0,005407794875114999)$ = 0,4986480546

Pada perhitungan di atas, hasil *output neuron* adalah 0,4986480546 yang akan diklasifikasikan ke kelas 0 atau tidak terkena penyakit stroke. Jika hasil *output neuron* lebih kecil dari 0,5 , maka data tersebut masuk ke dalam kelas 0, yaitu tidak terkena penyakit stroke, namun jika hasil *output* menunjukkan angka lebih besar sama dengan 0,5, maka data tersebut masuk ke dalam kelas 1, yang artinya terkena penyakit stroke.

3.6 Perhitungan *Deep Neural Network dengan Dropout*

Arsitektur yang akan digunakan dalam *neural network* dibagi menjadi 3 *layer* yang ditambah dengan *dropout* antara tiap *layer*, yaitu:

1. *S*atu buah input layer yang terdiri dari 10 *neuron* yang merepresentasikan 10 fitur dan menggunakan *activation function* tanh.
2. *S*atu buah hidden layer yang terdiri dari 10 *neuron* yang menggunakan *activation function* tanh.
3. Di antara *hidden layer* dan *output layer*, diberikan *dropout* dengan *rate* = 0,5.
4. Satu buah *output layer* yang terdiri dari 1 *neuron* dan menggunakan *activation function sigmoid*.

Gambar arsitektur *Deep Neural Network* dengan *dropout* dapat dilihat pada Gambar 2.8.

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 120 data secara *random* dengan rentang tertentu.

Inisialisasi Data:
Jumlah *input layer* = 1
Jumlah *neuron* pada *input layer* = 10
Activation function pada *input layer* = tanh

Dropout rate = 0,5

Jumlah *hidden layer* = 1
Jumlah *neuron* pada *hidden layer* = 10
Activation function pada *hidden layer* = tanh

Dropout rate = 0,5

Jumlah *output layer* = 1
Jumlah *neuron* pada *output layer* = 1
Activation function pada *output layer = sigmoid*

Rentang *weight* = -0,31622776601683794 sampai dengan 0,31622776601683794

Bias = 0

Input (weight setiap fitur yang ada):

$x1 = -0{,}02835948$ $x2 = -0{,}08527658$ $x3 = 0{,}26398902$ $x4 = -0{,}02807259$ $x5 = 0{,}04415979$ $x6 = -0{,}15166705$ $x7 = -0{,}18228751$ $x8 = 0{,}3134241$ $x9 = -0{,}00786456$ $x10 = 0{,}17467744$

Hidden layer:

$w_{k1}j_1 = -0{,}08154228$ $w_{k2}j_1 = 0{,}30040546$ $w_{k3}j_1 = 0{,}24854657$ $w_{k4}j_1 = 0{,}01691013$ $w_{k5}j_1 = -0{,}18283772$ $w_{k6}j_1 = 0{,}15720913$ $w_{k7}j_1 = 0{,}08201578$ $w_{k8}j_1 = -0{,}19598779$ $w_{k9}j_1 = 0{,}26268099$ $w_{k10}j_1 = 0{,}13676267$

$w_{k1}j_2 = 0{,}21293817$ $w_{k2}j_2 = -0{,}25220019$ $w_{k3}j_2 = 0{,}29390853$ $w_{k4}j_2 = -0{,}18997839$ $w_{k5}j_2 = -0{,}14565171$ $w_{k6}j_2 = 0{,}19445793$ $w_{k7}j_2 = -0{,}17048571$ $w_{k8}j_2 = -0{,}13914967$ $w_{k9}j_2 = -0{,}11090133$ $w_{k10}j_2 = 0{,}09844929$

$w_{k1}j_3 = -0{,}22548086$ $w_{k2}j_3 = 0{,}14510369$ $w_{k3}j_3 = 0{,}22841273$ $w_{k4}j_3 = -0{,}22168199$ $w_{k5}j_3 = 0{,}05429651$ $w_{k6}j_3 = -0{,}22866022$ $w_{k7}j_3 = -0{,}24450572$ $w_{k8}j_3 = -0{,}24502611$ $w_{k9}j_3 = -0{,}30133026$ $w_{k10}j_3 = -0{,}306759$

$w_{k1}j_4 = 0{,}03463625$ $w_{k2}j_4 = -0{,}00885326$ $w_{k3}j_4 = -0{,}22394442$ $w_{k4}j_4 = -0{,}06561425$ $w_{k5}j_4 = 0{,}07484731$ $w_{k6}j_4 = 0{,}01486269$ $w_{k7}j_4 = -0{,}10394584$ $w_{k8}j_4 = 0{,}17709923$ $w_{k9}j_4 = 0{,}07621706$ $w_{k10}j_4 = -0{,}09844928$

$w_{k1}j_5 = -0{,}12021486$ $w_{k2}j_5 = -0{,}07829361$ $w_{k3}j_5 = -0{,}12997544$ $w_{k4}j_5 = 0{,}21279237$ $w_{k5}j_5 = 0{,}20977342$ $w_{k6}j_5 = -0{,}22495591$ $w_{k7}j_5 = -0{,}00190682$ $w_{k8}j_5 = -0{,}22943178$ $w_{k9}j_5 = 0{,}27366045$ $w_{k10}j_5 = -0{,}20595517$

$w_{k1}j_6 = -0{,}00716274$ $w_{k2}j_6 = -0{,}05448304$ $w_{k3}j_6 = 0{,}06476339$ $w_{k4}j_6 = 0{,}03532464$ $w_{k5}j_6 = 0{,}09245855$ $w_{k6}j_6 = 0{,}1695172$ $w_{k7}j_6 = 0{,}23680192$ $w_{k8}j_6 = 0{,}04477918$ $w_{k9}j_6 = 0{,}24950235$ $w_{k10}j_6 = -0{,}30254995$

$w_{k1}j_7 = -0{,}14745197$ $w_{k2}j_7 = -0{,}08891625$ $w_{k3}j_7 = 0{,}22152022$ $w_{k4}j_7 = -0{,}0214877$ $w_{k5}j_7 = -0{,}12194666$ $w_{k6}j_7 = -0{,}05346982$ $w_{k7}j_7 = 0{,}26132289$ $w_{k8}j_7 = 0{,}26464311$ $w_{k9}j_7 = 0{,}29086091$ $w_{k10}j_7 = 0{,}07891517$

$w_{k1}j_8 = 0{,}30138627$ $w_{k2}j_8 = -0{,}17017802$ $w_{k3}j_8 = 0{,}0968307$ $w_{k4}j_8 = 0{,}26289768$ $w_{k5}j_8 = 0{,}01635267$ $w_{k6}j_8 = -0{,}06321122$ $w_{k7}j_8 = 0{,}1968099$ $w_{k8}j_8 = -0{,}14343715$ $w_{k9}j_8 = -0{,}01907697$ $w_{k10}j_8 = -0{,}24814027$

$w_{k1}j_9 = 0{,}30255215$ $w_{k2}j_9 = -0{,}08199351$ $w_{k3}j_9 = 0{,}17765137$ $w_{k4}j_9 = 0{,}07950517$ $w_{k5}j_9 = 0{,}15693231$ $w_{k6}j_9 = -0{,}06841941$ $w_{k7}j_9 = -0{,}01554272$ $w_{k8}j_9 = 0{,}21543019$ $w_{k9}j_9 = -0{,}19608282$ $w_{k10}j_9 = 0{,}27772592$

$w_{k1}j_{10} = -0{,}27114336$ $w_{k2}j_{10} = -0{,}12175201$ $w_{k3}j_{10} = 0{,}03013474$ $w_{k4}j_{10} = -0{,}10994942$ $w_{k5}j_{10} = 0{,}10416485$ $w_{k6}j_{10} = 0{,}21983722$ $w_{k7}j_{10} = 0{,}296758$ $w_{k8}j_{10} = 0{,}10953304$ $w_{k9}j_{10} = -0{,}06145034$ $w_{k10}j_{10} = 0{,}27154544$

Output layer:

$w_{k1}j_{11} = -0{,}14147132$ $w_{k2}j_{11} = 0{,}05554158$ $w_{k3}j_{11} = -0{,}27054289$ $w_{k4}j_{11} = -0{,}03277796$ $w_{k5}j_{11} = -0{,}10409342$ $w_{k6}j_{11} = -0{,}24334135$ $w_{k7}j_{11} = -0{,}06665223$ $w_{k8}j_{11} = 0{,}01262219$ $w_{k9}j_{11} = 0{,}18286435$ $w_{k10}j_{11} = 0{,}16453867$

Berikut ini merupakan perhitungan terhadap *hidden layer* dengan menggunakan persamaan 2.1 dan 2.3:

$h1 = ((w_{k1}j_1 * x1) + (w_{k2}j_1 * x2) + (w_{k3}j_1 * x3) + (w_{k4}j_1 * x4) + (w_{k5}j_1 * x5) + (w_{k6}j_1 * x6) + (w_{k7}j_1 * x7) + (w_{k8}j_1 * x8) + (w_{k9}j_1 * x9) + (w_{k10}j_1 * x10)) + b$

$h1$ = ((-0,08154228 * -0,02835948) + (-0,02835948 * -0,08527658) + (0,24854657 * 0,26398902) + (0,01691013 * -0,02807259) + (-0,18283772 * 0,04415979) + (0,15720913 * -0,15166705) + (0,08201578 * -0,18228751) + (-0,19598779 * 0,3134241) + (0,26268099 * -0,00786456) + (0,13676267 * 0,17467744)) + 0

$h1$ = -0,016602036215524193

$h2 = ((w_{k1}j_2 * x1) + (w_{k2}j_2 * x2) + (w_{k3}j_2 * x3) + (w_{k4}j_2 * x4) + (w_{k5}j_2 * x5) + (w_{k6}j_2 * x6) + (w_{k7}j_2 * x7) + (w_{k8}j_2 * x8) + (w_{k9}j_2 * x9) + (w_{k10}j_2 * x10)) + b$

$h2$ = ((0,21293817 * -0,02835948) + (-0,25220019 * -0,08527658) + (0,29390853 * 0,26398902) + (-0,18997839 * -0,02807259) + (-0,14565171 * 0,04415979) + (0,19445793 * -0,15166705) + (-0,17048571 * -0,18228751) + (-0,13914967 * 0,3134241) + (-0,11090133 * -0,00786456) + (0,09844929 * 0,17467744)) + 0

$h2$ = 0,06799857023423941

$h3 = ((w_{k1}j_3 * x1) + (w_{k2}j_3 * x2) + (w_{k3}j_3 * x3) + (w_{k4}j_3 * x4) + (w_{k5}j_3 * x5) + (w_{k6}j_3 * x6) + (w_{k7}j_3 * x7) + (w_{k8}j_3 * x8) + (w_{k9}j_3 * x9) + (w_{k10}j_3 * x10)) + b$

$h3$ = ((-0,22548086 * -0,02835948) + (0,14510369 * -0,08527658) + (0,22841273 * 0,26398902) + (-0,22168199 * -0,02807259) + (0,05429651 * 0,04415979) + (-0,22866022 * -0,15166705) + (-0,24450572 * -0,18228751) + (0,24502611 * 0,3134241) + (-0,30133026 * -0,00786456) + (-0,306759 * 0,17467744)) + 0

$h3$ = 0,16777353734936898

$h4 = ((w_{k1}j_4 * x1) + (w_{k2}j_4 * x2) + (w_{k3}j_4 * x3) + (w_{k4}j_4 * x4) + (w_{k5}j_4 * x5) + (w_{k6}j_4 * x6) + (w_{k7}j_4 * x7) + (w_{k8}j_4 * x8) + (w_{k9}j_4 * x9) + (w_{k10}j_4 * x10)) + b$

$h4$ = ((0,03463625 * -0,02835948) + (-0,00885326 * -0,08527658) + (-0,22394442 * 0,26398902) + (-0,06561425 * -0,02807259) + (0,07484731 * 0,04415979) + (0,01486269 * -0,15166705) + (-0,10394584 * -0,18228751) + (0,17709923 * 0,3134241) + (0,07621706 * -0,00786456) + (0,09844928 * 0,17467744)) + 0

$h4$ = 0,03459951448869128

$h5 = ((w_{k1}j_5 * x1) + (w_{k2}j_5 * x2) + (w_{k3}j_5 * x3) + (w_{k4}j_5 * x4) + (w_{k5}j_5 * x5) + (w_{k6}j_5 * x6) + (w_{k7}j_5 * x7) + (w_{k8}j_5 * x8) + (w_{k9}j_5 * x9) + (w_{k10}j_5 * x10)) + b$

$h5$ = ((-0,12021486 * -0,02835948) + (-0,07829361 * -0,08527658) + (-0,12997544 * 0,26398902) + (0,21279237 * -0,02807259) + (0,20977342 * 0,04415979) + (-0,22495591 * -0,15166705) + (-0,00190682 * -0,18228751) + (-0,22943178 * 0,3134241) + (0,27366045 * -0,00786456) + (-0,20595517 * 0,17467744)) + 0

$h5$ = -0,09650773091582979

$h6 = ((w_{k1}j_6 * x1) + (w_{k2}j_6 * x2) + (w_{k3}j_6 * x3) + (w_{k4}j_6 * x4) + (w_{k5}j_6 * x5) + (w_{k6}j_6 * x6) + (w_{k7}j_6 * x7) + (w_{k8}j_6 * x8) + (w_{k9}j_6 * x9) + (w_{k10}j_6 * x10)) + b$

$h6 = ((-0,00716274 * -0,02835948) + (-0,05448304 * -0,08527658) + (0,06476339 * 0,26398902) + (0,03532464 * -0,02807259) + (0,09245855 * 0,04415979) + (0,1695172 * -0,15166705) + (0,23680192 * -0,18228751) + (0,04477918 * 0,3134241) + (0,24950235 * -0,00786456) + (-0,30254995 * 0,17467744)) + 0$

$h6 = -0,08461482998284209$

$h7 = ((w_{k1}j_7 * x1) + (w_{k2}j_7 * x2) + (w_{k3}j_7 * x3) + (w_{k4}j_7 * x4) + (w_{k5}j_7 * x5) + (w_{k6}j_7 * x6) + (w_{k7}j_7 * x7) + (w_{k8}j_7 * x8) + (w_{k9}j_7 * x9) + (w_{k10}j_7 * x10)) + b$

$h7 = ((-0,14745197 * -0,02835948) + (-0,08891625 * -0,08527658) + (0,22152022 * 0,26398902) + (0,0214877 * -0,02807259) + (-0,12194666 * 0,04415979) + (-0,05346982 * -0,15166705) + (0,26132289 * -0,18228751) + (0,26464311 * 0,3134241) + (0,29086091 * -0,00786456) + (0,07891517 * 0,17467744)) + 0$

$h7 = 0,1191711327063339$

$h8 = ((w_{k1}j_8 * x1) + (w_{k2}j_8 * x2) + (w_{k3}j_8 * x3) + (w_{k4}j_8 * x4) + (w_{k5}j_8 * x5) + (w_{k6}j_8 * x6) + (w_{k7}j_8 * x7) + (w_{k8}j_8 * x8) + (w_{k9}j_8 * x9) + (w_{k10}j_8 * x10)) + b$

$h8 = ((0,30138627 * -0,02835948) + (-0,17017802 * -0,08527658) + (0,0968307 * 0,26398902) + (0,26289768 * -0,02807259) + (0,01635267 * 0,04415979) + (-0,06321122 * -0,15166705) + (0,1968099 * -0,18228751) + (-0,14343715 * 0,3134241) + (-0,01907697 * -0,00786456) + (-0,24814027 * 0,17467744)) + 0$

$h8 = -0,0895708672147945$

$h9 = ((w_{k1}j_9 * x1) + (w_{k2}j_9 * x2) + (w_{k3}j_9 * x3) + (w_{k4}j_9 * x4) + (w_{k5}j_9 * x5) + (w_{k6}j_9 * x6) + (w_{k7}j_9 * x7) + (w_{k8}j_9 * x8) + (w_{k9}j_9 * x9) + (w_{k10}j_9 * x10)) + b$

$h9 = ((0,30255215 * -0,02835948) + (-0,08199351 * -0,08527658) + (0,17765137 * 0,26398902) + (0,07950517 * -0,02807259) + (0,15693231 * 0,04415979) + (-0,06841941 * -0,15166705) + (-0,01554272 * -0,18228751) + (-0,21543019 * 0,3134241) + (-0,19608282 * -0,00786456) + (0,27772592 * 0,17467744)) + 0$

$h9 = 0,0457518555709885$

$h10 = ((w_{k1}j_{10} * x1) + (w_{k2}j_{10} * x2) + (w_{k3}j_{10} * x3) + (w_{k4}j_{10} * x4) + (w_{k5}j_{10} * x5) + (w_{k6}j_{10} * x6) + (w_{k7}j_{10} * x7) + (w_{k8}j_{10} * x8) + (w_{k9}j_{10} * x9) + (w_{k10}j_{10} * x10)) + b$

$h10 = ((-0,27114336 * -0,02835948) + (-0,12175201 * -0,08527658) + (0,03013474 * 0,26398902) + (-0,10994942 * -0,02807259) + (0,10416485 * 0,04415979) + (0,21983722 * -0,15166705) + (0,296758 * -0,18228751) + (0,10953304 * 0,3134241) + (-0,06145034 * -0,00786456) + (0,27154544 * 0,17467744)) + 0$

$h10 = 0,028522880227219695$

Setelah mendapatkan hasil perhitungan terhadap *hidden layer*, selanjutnya diterapkan *activation function tanh* sesuai persamaan 2.5.

$tanh\ (h1) = \frac{e^{h1} - e^{-h1}}{e^{h1} + e^{-h1}}$

$tanh\ (h1) = \frac{e^{-0,016602036215524193} - e^{--0,016602036215524193}}{e^{-0,016602036215524193} + e^{--0,016602036215524193}}$

$tanh\ (h1) = -0,0166005$

$tanh\ (h2) = \frac{e^{h2} - e^{-h2}}{e^{h2} + e^{-h2}}$

$tanh\ (h2) = \frac{e^{0,06799857023423941} - e^{-0,06799857023423941}}{e^{0,06799857023423941} + e^{-0,06799857023423941}}$

$tanh\ (h2) = 0,067894$

$tanh\ (h3) = \frac{e^{h3} - e^{-h3}}{e^{h3} + e^{-h3}}$

$tanh\ (h3) = \frac{e^{0,16777353734936898} - e^{-0,16777353734936898}}{e^{0,16777353734936898} + e^{-0,16777353734936898}}$

$tanh\ (h3) = 0,166217$

$tanh\ (h4) = \frac{e^{h4} - e^{-h4}}{e^{h4} + e^{-h4}}$

$tanh\ (h4) = \frac{e^{0,03459951448869128} - e^{-0,03459951448869128}}{e^{0,03459951448869128} + e^{-0,03459951448869128}}$

$tanh\ (h4) = 0,0345857$

$tanh\ (h5) = \frac{e^{h5} - e^{-h5}}{e^{h5} + e^{-h5}}$

$tanh\ (h5) = \frac{e^{-0,09650773091582979} - e^{--0,09650773091582979}}{e^{-0,09650773091582979} + e^{--0,09650773091582979}}$

$tanh\ (h5) = -0,0962092$

$tanh\ (h6) = \frac{e^{h6} - e^{-h6}}{e^{h6} + e^{-h6}}$

$tanh\ (h6) = \frac{e^{-0,08461482998284209} - e^{--0,08461482998284209}}{e^{-0,08461482998284209} + e^{--0,08461482998284209}}$

$tanh\ (h6) = -0,0844135$

$tanh\ (h7) = \frac{e^{h7} - e^{-h7}}{e^{h7} + e^{-h7}}$

$tanh\ (h7) = \frac{e^{0,1191711327063339} - e^{-0,1191711327063339}}{e^{0,1191711327063339} + e^{-0,1191711327063339}}$

$tanh\ (h7) = 0,11861$

$tanh\ (h8) = \frac{e^{h8} - e^{-h8}}{e^{h8} + e^{-h8}}$

$tanh\ (h8) = \frac{e^{-0,0895708672147945} - e^{--0,0895708672147945}}{e^{-0,0895708672147945} + e^{--0,0895708672147945}}$

$tanh\ (h8) = -0,0893321$

$tanh\ (h9) = \frac{e^{h9} - e^{-h9}}{e^{h9} + e^{-h9}}$

$tanh\ (h9) = \frac{e^{0,0457518555709885} - e^{-0,0457518555709885}}{e^{0,0457518555709885} + e^{-0,0457518555709885}}$

$tanh\ (h9) = 0,04572$

$tanh\ (h10) = \frac{e^{h10} - e^{-h10}}{e^{h10} + e^{-h10}}$

$tanh\ (h10) = \frac{e^{0,028522880227219695} - e^{-0,028522880227219695}}{e^{0,028522880227219695} + e^{-0,028522880227219695}}$

$tanh\ (h10) = 0,0285151$

Setelah mendapatkan hasil perhitungan *activation function tanh*, dilakukan perhitungan *dropout* sesuai persamaan 2.8:

$dropout_2$ = rate * tanh($h2$)

$dropout_2$ = 0,5 * 0,067894

$dropout_2$ = 0,033947

$dropout_3$ = rate * tanh($h3$)

$dropout_3$ = 0,5 * 0,166217

$dropout_3$ = 0,0831085

$dropout_4$ = rate * tanh($h4$)

$dropout_4$ = 0,5 * 0,0345857

$dropout_4$ = 0,01729285

$dropout_7$ = rate * tanh($h7$)

$dropout_7$ = 0,5 * 0,11861

$dropout_7$ = 0,059305

$dropout_9$ = rate * tanh($h9$)

$dropout_9$ = 0,5 * -0,04572

$dropout_9$ = -0,02286

Selanjutnya, dilakukan perhitungan terhadap *output layer* dengan menggunakan persamaan 2.1 dan 2.3:

$output$ = (($w_{k2} j_1 1$ * $dropout_2$) + ($w_{k3} j_1 1$ * $dropout_3$) + ($w_{k4} j_1 1$ * $dropout_4$) + ($w_{k7} j_1 1$ * $dropout_7$) + ($w_{k9} j_1 1$ * $dropout_9$)) + b

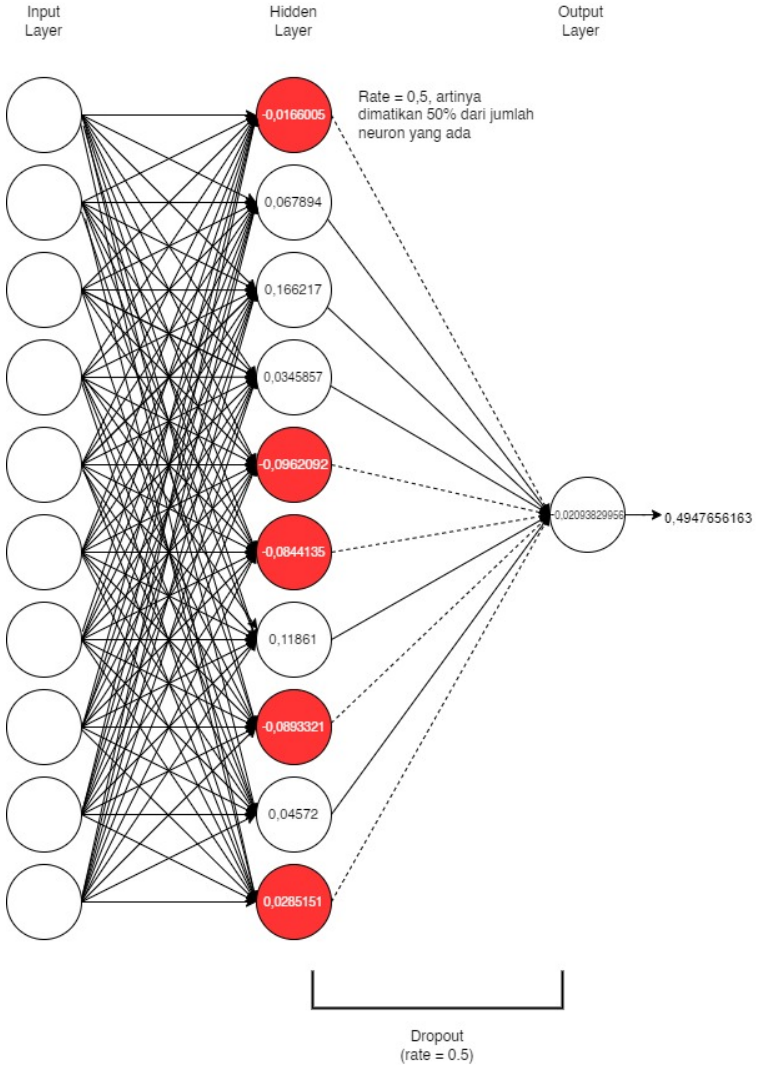
$output$ = ((0,05554158 * 0,033947) + (-0,27054289 * 0,0831085) + (-0,03277796 * 0,01729285) + (-0,06665223 * 0,059305) + (0,18286435 * -0,02286) + 0)

$output$ = -0,02093829956

Langkah terakhir, adalah melakukan perhitungan dengan *activation function sigmoid* pada *output layer* dengan menggunakakan persamaan 2.4 untuk mendapatkan hasil prediksi antara 0 (tidak stroke) atau 1 (stroke):

$\sigma(z) = = 1/(1+exp(-z))$
$\sigma(-0,02093829956) = 1 / (1 + \exp(- -0,02093829956))$
$\sigma(-0,02093829956 = 0,4947656163$

Agar lebih jelas, arsitektur DNN *dropout* sampai perhitungan selesai dapat dilihat pada Gambar 3.14 di bawah ini.



Gambar 3.14 *Neuron* yang dimatikan pada model DNN dan *dropout*

Pada perhitungan di atas, hasil *output neuron* adalah 0,4947656163 yang akan diklasifikasikan ke kelas 0 atau tidak terkena penyakit stroke. Jika hasil *output*

neuron lebih kecil dari 0,5 , maka data tersebut masuk ke dalam kelas 0, yaitu tidak terkena penyakit stroke, namun jika hasil *output* menunjukkan angka lebih besar sama dengan 0,5, maka data tersebut masuk ke dalam kelas 1, yang artinya terkena penyakit stroke.

3.7 Perhitungan Cost untuk Cost-Sensitive Learning

Sesuai dengan persamaan yang sudah dijelaskan pada Persamaan 2.11, maka hasil *output* dari perhitungan *Deep Neural Network* dengan *dropout* akan digunakan untuk mencari *cost*.

$$O_i^*(x) = \eta \sum_{j=1}^{M} O_i(x) C(i,j)$$
$$O_i^*(x) = \eta(0,4947656163 * (42336/781))$$
$$O_i^*(x) = \eta 0,4947656163 * 54,207426$$
$$O_i^*(x) = \eta 26,81997053$$
$$O_i^*(x) = 1$$

Pada perhitungan di atas, dikarenakan *output neural network* yang dipakai hanya 1 buah, maka hasil *cost* dari kesalahan klasifikasi jika dinormalisasi pasti akan menghasilkan angka 1. Jika *output neural network* lebih dari 1, maka hasil normalisasi akan lebih terlihat. Hasil *cost* ini akan dibandingkan dengan *output* dari *weight neural network* lainnya dan akan diambil nilai yang terbesar sebagai nilai *cost-sensitive learning*.